TU Clausthal

Clausthal University of Technology

Fast Adaptive Silhouette Area based Template Matching

Daniel Mohr and Gabriel Zachmann

IfI Technical Report Series IfI-10-04

Department of Informatics
Clausthal University of Technology

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Michael Köster
Contact: michael.koester@tu-clausthal.de
URL: http://www.in.tu-clausthal.de/forschung/technical-reports/
ISSN: 1860-8477

The IfI Review Board

Fast Adaptive Silhouette Area based Template Matching

Daniel Mohr and Gabriel Zachmann

Clausthal University of Technology, Department of Informatics, Germany
{dmoh,zach}@tu-clausthal.de

Abstract

Template matching is a well-proven approach in the area of articulated object tracking. Matching accuracy and computation time of template matching are essential and yet often conflicting goals.
In this paper, we present a novel, adaptive template matching approach based on the silhouette area of the articulated object. With our approach, the ratio between accuracy and speed simply is a modifiable parameter, and, even at high accuracy, it is still faster than a state-of-the-art approach.
We approximate the silhouette area by a small set of axis-aligned rectangles. Utilizing the integral image, we can thus compare a silhouette with an input image at an arbitrary position independently of the resolution of the input image. In addition, our rectangle covering yields a very memory efficient representation of templates.
Furthermore, we present a new method to build a template hierarchy optimized for our rectangular representation of template silhouettes.
With the template hierarchy, the complexity of our matching method for n templates is $O(\log n)$ and independent of the input resolution. For example, a set of 3000 templates can be matched in 2.3 ms.
Overall, our novel methods are an important contribution to a complete system for tracking articulated objects.

1 Introduction

Tracking an articulated object is a challenging task, especially, if the configuration space of the object has many degrees of freedom (DOF), e.g., the human hand has about 26 DOF. Most tracking approaches require the object to be in a predefined state. If this is not desired or impractical, one has to search the whole configuration space at initialization time. Such a global search, of course, only needs to find a coarse object state, which typically consists of two parts. The first part is a similarity measure between the object in each

configuration and the observed object (e.g. images from a camera). The second part is an algorithm to combine the best matching configurations, detect and eliminate false positives, potentially reduce the search space, and estimate the final object state. The focus of this paper is the first part of such a global search method.

Tracking articulated objects, there is a large number of object configurations that have to be compared with an input image. Therefore, this comparison should be extremely fast. We propose a novel method for very fast approximate area silhouette comparison between model templates and input images. For one comparison, Stenger et al. [Stenger et al., 2006] achieved a computation time proportional to the contour length of the template silhouette. We propose a new method, which reduces the computation time to be *independent* of the contour length and image resolution. It only depends on the desired accuracy of the template representation. This accuracy is a freely adjustable parameter in our approach. To achieve this, we first approximate all template silhouettes by axis-aligned rectangles, which is done in a preprocessing step. In the online phase, we compute the integral image of the segmented input image [Crow, 1984, Viola and Jones, 2001]. With this, the joint probability of a rectangle to match an image region can be computed by four lookups in the integral image. Moreover, we present an algorithm to build a template hierarchy that can compare a large set of templates in sublinear time.

Our *main contributions* are:

1. An algorithm that computes a representation of arbitrary shapes by a small set of axis-aligned rectangles with adjustable accuracy. This results in a resolution-independent, very memory efficient shape representation.

2. An algorithm to compare an object silhouette in $O(1)$. In contrast the algorithm proposed by [Stenger et al., 2006] needs $O(\text{contour length})$.

3. We propose an algorithm to cluster templates hierarchically guided by their mutually overlapping areas. This hierarchy further reduces the matching complexity for n templates from $O(n)$ to $O(\log n)$.

We assume that the object to be tracked can be segmented in the input image. Mostly, background subtraction or color segmentation approaches are used to achieve this. The result of a segmentation is a probability map, which supplies for each pixel the probability that this pixel belongs to the foreground (target object) or background. For our approach, there is no need to binarize this probability map.

It should be obvious that our proposed methods are suitable for any kind of objects. For sake of clarity, though, we will describe our novel methods in the following by the example of the human hand, since human hand tracking is our long-term goal. This includes the full 26 DOFs of the hand, not

only a few poses. To achieve this challenging task, we mainly use two different features for matching: edge gradients and skin color. In this paper, we focus on the skin color feature. We use a skin segmentation algorithm that computes for each image pixel the probability to represent skin or background, respectively. We generate our templates by an artificial 3D hand model. This model can be rendered in any desired state, and it can be easily projected onto 2D and binarized to get the hand silhouette. Given an input image, the goal then is to find the best matching hand silhouette.

We use the joint probability as proposed by Stenger et. al [Stenger et al., 2006] to compare the silhouettes with the segmentation. A simple area overlap, of course, could be used, too. The only difference is that the sum instead of the product of probabilities would have been computed. For details, see Sec. 3.

2 Related Work

A lot of object tracking approaches based on silhouette comparison have been proposed. The approaches can be divided into two classes. The first class needs a binary silhouette of both, the model and the query image. The second class compares binary model silhouette area with the likelihood map of the query image.

A simple method belonging to the first class is used in [Lin et al., 2004, Wu et al., 2001]. The difference between the model silhouette and segmented foreground area in the query image is computed. The exponential of the negative squared difference is used as silhouette matching probability. A slightly different measure is used by Kato et. al [Kato et al., 2006]. First, they define the model silhouette area A_M, the segmented area A_I and the intersecting area $A_O = A_I \cap A_M$. The differences $A_I - A_O$, $A_M - A_O$ and $A_I - A_M$ are integrated in the same way, as described above, into the overall measure. In [Ouhaddi and Horain, 1999], the non-overlapping area of the model and the segmented silhouettes are integrated into classical optimization methods, e.g. Levenberg-Marquardt or downhill simplex. Nirei et. al [Nirei et al., 1996] first compute the distance transform of both the input and model silhouette. Regarding the distance transformed images as vectors, they compute the normalized scalar product of these vectors. Additionally, the model is divided into meaningful parts. Next, for each part, the area overlap between the part and the segmented input image is computed. Then, a weighted sum of the quotient between this overlap and the area of the corresponding model part is computed. The final similarity is the sum of the scalar product and the weighted sum. In [Amai et al., 2004, Shimada et al., 2001] a compact description of the hand model is generated. Vectors from the gravity center to sample points on the silhouette boundary, normalized by the square root of the silhouette area, are used as hand representation. During tracking, the same transformations are performed to the binary input image and the

vector is compared to the database. A completely different approach is proposed by Zhou and Huang [Zhou and Huang, 2005]. Although they extract the silhouette from the input image, they use only local features extracted from the silhouette boundary. Their features are inspired by the SIFT descriptor [Lowe, 1999]. Each silhouette is described by a set of feature points. The chamfer distance between the feature points is used as similarity measure.

All the aforementioned approaches have the same drawback: to ensure that the algorithms work, a binary segmentation of the input image of high quality is necessary. The thresholds, needed for the binarization, are often not easy to determine.

To our knowledge, there are much less approaches working directly on the color likelihood map of a segmentation. In [Zhou and Huang, 2003] the skin-color likelihood is used. For further matching, new features, called likelihood edges, are generated by applying an edge operator to the likelihood ratio image. But, in many cases, this leads to a very noisy edge image. In [Stenger et al., 2006, Stenger, 2004, Sudderth et al., 2004], the skin-color likelihood map is directly compared with hand silhouettes. The product of all skin probabilities at the silhouette foreground is multiplied with the product of all background probabilities in the template background. Stenger et. al [Stenger, 2004] proposed a method for the efficient computation of this joint probability. The row-wise prefix sum in the log-likelihood image is computed. The original product along all pixels in a row reduces to three lookups in the prefix sum. Thus, the complexity to compute the joint probability is linear in the number of pixels along the template border.

Nevertheless, the above mentioned approach has some disadvantages. First of all, the template representation is resolution dependent. Typically, the distance of the object from the camera is not constant, and thus different sizes of the templates need to be considered. Consequently, for each scale, an extra set of the templates has to be kept in memory. Also, the higher the resolution of the images, the higher is the matching cost.

Our approach does not have all these disadvantages.

3 Silhouette representation

In the rest of this paper, we will denote the *template silhouette* simply as *template*. To avoid the issues mentioned in the previous section, we propose a novel *resolution-independent representation* of templates, which is the key to our fast matching approach. We propose to approximate a template by a set of axis-aligned rectangles. With such a representation, one can perform template matching at arbitrary resolutions in constant time with respect to the template size. Figure 1 shows an overview of our approach.

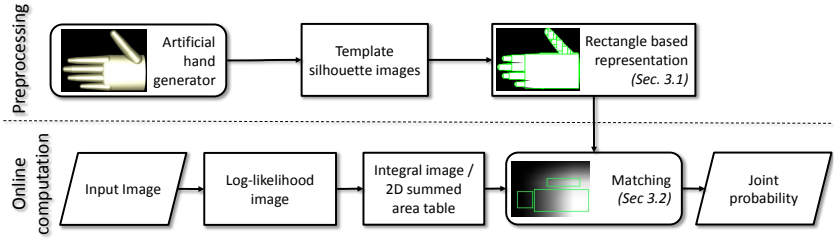

Figure 1: Overview of our approach using rectangle sets to approximate a silhouette. This speeds up the matching by a factor 5–30 compared to the approach proposed by Stenger et. al [Stenger et al., 2006].

We denote the integral image of a gray scale image I by II:

$$\mathit{II}(x,y) = \sum_{\substack{0\le i\le x\\ 0\le j\le y}} I(i,j) \tag{1}$$

Let R be an axis-aligned rectangle with upper left corner $\mathbf{u}$ and lower right corner $\mathbf{v}$, both inside I. The sum of the area R of all pixels in I is given by

$$\sum_R I(i,j) = \mathit{II}(\mathbf{v}_x,\mathbf{v}_y) + \mathit{II}(\mathbf{u}_x-1,\mathbf{u}_y-1) - \mathit{II}(\mathbf{v}_x,\mathbf{u}_y-1) - I(\mathbf{u}_x-1,\mathbf{v}_y) \tag{2}$$

Let T with $T(x,y)\in\{0,1\}$ be a binary image representing a template. Let S and $\bar{S}$ denote the set of foreground and background pixels in T, resp. We compute a set of n mutually non-overlapping rectangles $\mathcal{R}=\{R_i\}_{i=1\cdots n}$ that cover S.

3.1 Rectangle Covering Computation

In the following, we denote a set of rectangles approximating S with $\mathcal{R}_S$. To obtain a good approximation, we minimize the symmetric difference of S and $\mathcal{R}_S$:

$$A = \min_{\mathcal{R}_S} \Big| (S \setminus \bigcup_{R_i\in\mathcal{R}_S} R_i) \cup (\bigcup_{R_i\in\mathcal{R}_S} R_i \setminus S) \Big| . \tag{3}$$

Obviously, there is a trade-off between A and $n=|\mathcal{R}_S|$: the smaller the number of rectangles, the faster the matching is, but also the more inaccurate. We can utilize this to obtain an adaptive template representation.

There is a large body of work solving similar problems. One has to differentiate between rectangle covering [Kumar and Ramesh, 1999, Wu and Sahni, 1990, Heinrich-Litan and Lübecke, 2006] and partitioning problems [Liou et al., 1990,

O'Rourke and Tewari, 2001]: covering allows an arbitrary overlap among the rectangles in $\mathcal{R}_S$ while partitioning does not. Most covering and partitioning algorithms compute solutions under the constraint that the rectangles lie completely inside the polygon to be covered. Our problem is similar to standard partitioning in that overlaps between the rectangles in $\mathcal{R}_S$ is undesired (in partitioning problems, it is not allowed at all), but it differs from partitioning because we can allow rectangles to cover a small part of the template background $\bar{S}$ or not cover small foreground regions, too. In fact, we even encourage a solution with above described small "errors" so as to keep the number of rectangles at a minimum. The reason is that, in the real world, S never perfectly matches the observed real hand anyway. Therefore, we can allow $A > 0$, which usually leads to solutions with much smaller numbers of rectangles in $\mathcal{R}_S$. To our knowledge, no algorithm has been presented that computes such an approximate rectangle covering. In the following, we present a simple and fast algorithm to obtain a solution for $A < \delta$.

First, the model (here, the human hand) is rendered at a given state, rasterized at a high resolution, and, after thresholding, a binary image T is obtained. We propose a greedy algorithm to compute the rectangle based template representation $\mathcal{R}_S \subset \mathcal{R}(T)$, where $\mathcal{R}(T)$ denotes the set of all rectangles in the image T. For an axis-aligned rectangle $R \in \mathcal{R}(T)$ we define its benefit:

$$F(R) = \sum_{\mathbf{x} \in R} (T(\mathbf{x}) - \tau) \tag{4}$$

The parameter $\tau \in [0, 1]$ allows us to control the trade-off between *covering background regions* and *not covering foreground regions*. It controls the penalty for background pixels covered by a rectangle in the solution $\mathcal{R}_S$.

We initialize $\mathcal{R}_S$ with the empty set. At each iteration j in the greedy algorithm, we try to find $R_j^{\text{opt}} = \text{argmax}_{R \subset \mathcal{R}(T)} F(R)$. Because we have no knowledge about R_j^{opt}, we use a two-step search strategy to estimate this rectangle. Our search strategy basically works as follows:

The *first step* is a recursive search. For a rectangle/image X, we define the function

$$M(X) = \text{argmax}\{F(R) \mid R \in \mathcal{R}(X) \text{ with size } (\frac{\text{width}(X)}{2}, \frac{\text{height}(X)}{2})\} \tag{5}$$

At recursion level 0, we compute $R_{\text{max}}^0 = M(T)$. At recursion levels $i > 0$, we compute $R_{\text{max}}^i = M(R_{\text{max}}^{i-1})$. We stop at recursion level k, if R_{max}^k is completely inside the foreground of T.

In the *second step*, we optimize the size of the rectangle R_{max}^k. This is done by simply moving the rectangle borders so long as $F(R_{\text{max}}^k)$ grows. We obtain the final rectangle R_j^{opt}. Note that τ influences the optimization result. The higher τ is, the more covering a background pixel will be penalized. For example, if $\tau = 1$, then R_{max}^i will never cover any background pixel.

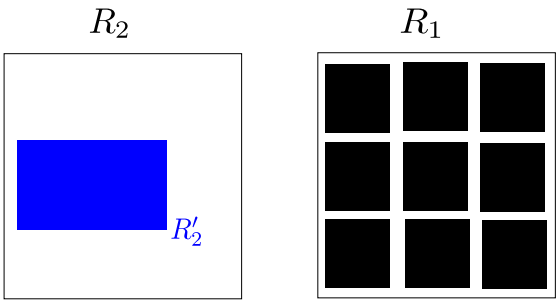

Figure 2: Simplified case to illustrate the necessity for *step three* in our covering algorithm, which basically detects ill posed regions and temporarily disables them for the *first step*.

We add the rectangle R_j^{opt} to our final solution $\mathcal{R}_S = \mathcal{R}_S \cup \{R_j^{\text{opt}}\}$. The region R_j^{opt} is then erased from the foreground in T and the *first* and *second* step are repeated until the desired covering accuracy $A < \delta$ from Eq. 3 is achieved. The algorithm in this form, however, has one problem. Consider a configuration as shown in Figure 2. There is a rectangle R_1 that contains a large number of small foreground regions, i.e. $F(R_1)$ is large and another rectangle R_2, with $F(R_2) < F(R_1)$, but that contains just one fairly large region, which itself can contain a rectangle R_2'. It can happen that the area of R_2' is larger than any of the foreground regions in R_1. However, the algorithm so far would still choose R_1 first.

To overcome this problem we extend the greedy algorithm by a *third* step. We test whether the rectangle R_j^{opt} from the *second* step is larger than a threshold r. If the test fails, we do not add the rectangle to $\mathcal{R}_S$ and further disable the region R_j^{opt} for the following iterations. If, at any time, the whole image T is disabled for search, all disabled search regions are enabled again for searching and r is set to the size of the largest rectangle found by the recursive search in the *first* step. There is one problem left: the initialization of the threshold r. We set it to $r = \text{size}(T)$. The reason is, that we have the demand that our algorithm works for all cases, even if the template T contains only foreground pixels. If r would be set smaller, we could not obtain the optimal solution in this case, which, of course, is one rectangle, covering the whole template image. Example coverings computed by the algorithm are shown in Figure 3.

F can be evaluated by four lookups in the integral image IT of T (see Eq. 2), which can be precomputed at initialization. Let w and h be the width and height of T then the complexity of the *first step* can be described by the fol-

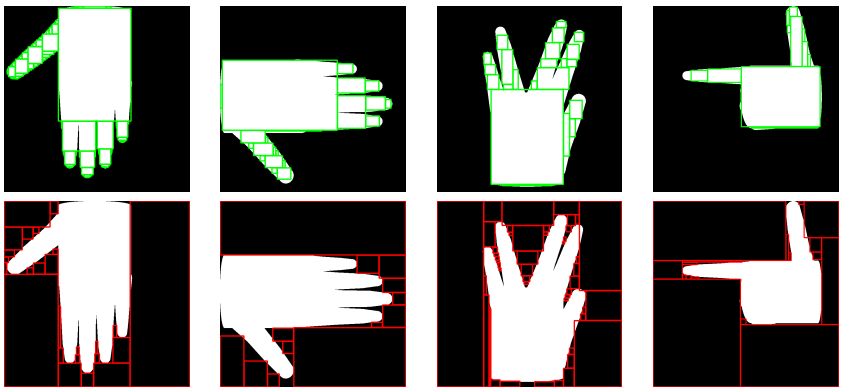

Figure 3: Example silhouettes approximated by a set of rectangles. The *upper* row shows rectangles approximating the foreground, the *lower* one the rectangles approximating the background.

lowing recursive formula:

$$T(w \cdot h) = c \cdot \frac{w}{2} \cdot \frac{h}{2} + T(\frac{w}{2} \cdot \frac{h}{2}) \tag{6}$$

which is in $O(w \cdot h)$. It is trivial to see that the complexity of the *second step* is $O(w \cdot h)$. The update cost of $I\!I$ after erasing R_j^{opt} in T is also linear in the size of T. Figure 4 illustrates the regions that need to be updated. Therefore, the overall complexity of our rectangle covering algorithm is $O(|\mathcal{R}_S| \cdot w \cdot h)$. For *step two*, we also tried the well-known Nelder-Mead optimization (Numerical Recipes implementation). In our experience, however, the quality of the resulting rectangles was never better and sometimes much worse, while the computation time was about a factor *1000* higher.

3.2 Matching Templates

In the previous section, we have developed an algorithm to compute for each template a resolution-independent compact representation consisting of axis-aligned rectangles. In the following, this representation will be used for fast template matching.

Our goal is to compare a template S with an input image I at a given position $\mathbf{p}$ using the joint probability (see Stenger et. al [Stenger, 2004]). The first step is the foreground/background segmentation. We use the color likelihood instead of the binary segmentation due to its higher robustness against noise and imperfect segmentation. In the following, the color likelihood image of an input image I is denoted with $\tilde{L}$ with $\tilde{L}(x, y) \in [0, 1]$. To convert the

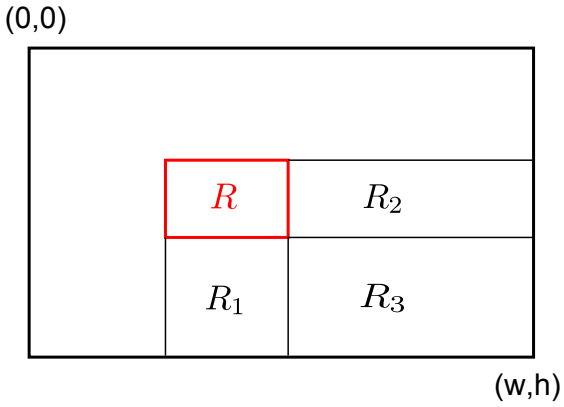

Figure 4: Regions in an integral image that have to be updated after deleting rectangle R: The origin is at the top left corner. Only the regions consisting of R and R_1, R_2 and R_3, which are on the right and/or bottom of R are affected (see Eq. 1.

product in the joint probability into sums, we take the pixel-wise logarithm: $L(x, y) = \log \tilde{L}(x, y)$.

Utilizing Eq. 2, we can compute the joint probability at position $\mathbf{p}$ by:

$$P_S(\mathbf{p}) = \sum_{R_i \in \mathcal{R}_S} (IL(\begin{pmatrix} \mathrm{v}_x^i \\ \mathrm{v}_y^i \end{pmatrix} + \mathbf{p}) + IL(\begin{pmatrix} \mathrm{u}_x^i \\ \mathrm{u}_y^i \end{pmatrix} + \mathbf{p}) - IL(\begin{pmatrix} \mathrm{v}_x^i \\ \mathrm{u}_y^i \end{pmatrix} + \mathbf{p}) - IL(\begin{pmatrix} \mathrm{u}_x^i \\ \mathrm{v}_y^i \end{pmatrix} + \mathbf{p})) \quad (7)$$

IL denotes the integral image of the log-likelihood image L. The rectangle set $\mathcal{R}_S$ approximates only the template foreground. To get the appropriate match probability for a template, one has to take into account the background distribution, too.

Fortunately, the set of background pixels $\bar{S}$ of a template image, obviously, can be approximated by a set of rectangles with the same algorithm described in the last section. Having computed $\mathcal{R}_{\bar{S}}$, we can compute $P_{\bar{S}}$.

P_S and $P_{\bar{S}}$ are resolution-dependent and need to be normalized.

In the following, we explain the normalization for P_S. $P_{\bar{S}}$ can be normalized analogously. A naïve approach is to normalize P_S by the number of actually matched pixels for a template. There are two reasons against this normalization method. The first one is that we want a "smart" matching at the border, i.e. when the template is partially outside the input image. The second reason is explained in Sec. 3.5.

For each pixel not covered by any rectangle, including *all* pixels of the template image that are outside the borders of the input image, we assume a likelihood value of $\frac{1}{2}$. The value is motivated by the assumption that for a pixel

not yet observed, the probability to be foreground or background is equal. Let us denote the number of pixels of rectangle R_i inside the input image at position $\mathbf{p}$ in an input image by $N^{\mathbf{p}}_{R_i}$. Then we normalize P_S as follows:

$$P^N_S(\mathbf{p}) = \frac{1}{|S|}\left(P_S(\mathbf{p}) + \log(\frac{1}{2})(|S| - N^{\mathbf{p}}_R)\right), \quad \text{with } N^{\mathbf{p}}_R = \sum_{R_i \in \mathcal{R}_S} N^{\mathbf{p}}_{R_i} \tag{8}$$

The normalized probability $P^N_{\bar{S}}$ of the background is calculated analog. The final joint probability is

$$P = \exp(\frac{1}{2}(P^N_S + P^N_{\bar{S}})) \tag{9}$$

where $P_{\bar{S}}$ is the background joint probability. Treating the joint probabilities for the foreground and background equally takes care of the fact that different template shapes have different area relative to their bounding box used in the template: in a template with fewer foreground pixels, the matching of the background pixels should not have a bigger weight than the foreground pixels and vice versa.

To determine the size of the target object one has to match the templates at different scales. This can be done easily by scaling the corner values for all rectangles accordingly. No additional representation has to be stored. Comparability between the same template at different sizes is ensured by the normalization.

3.3 The Template Hierarchy

In the previous section, we have described a novel method to match an arbitrary template T to an input image I. The complexity of this method is independent of the input image resolution and template size. In a typical tracking application, especially when dealing with articulated objects, a huge number of templates must be matched. At initialization, there is no previous knowledge about position and state of the target object. Thus, one has to scan a huge number of different templates. A suitable approach to reduce the complexity from $O(\#templates)$ to $O(\log \#templates)$ is to use a template hierarchy. However, building a well working one is still a challenging task.

In this section, we propose an approach to build a hierarchy that exploits our representation of the templates by rectangles. It utilizes information about spatial similarity of template silhouettes, not only distance measures between templates. This yields the great advantage that silhouette areas, shared by a set of templates, are matched only once and not separately for each template. This greatly reduces the computation time.

In many applications tracking an articulated object, one can observe that the intersection area of a number of templates is fairly big. Therefore, one

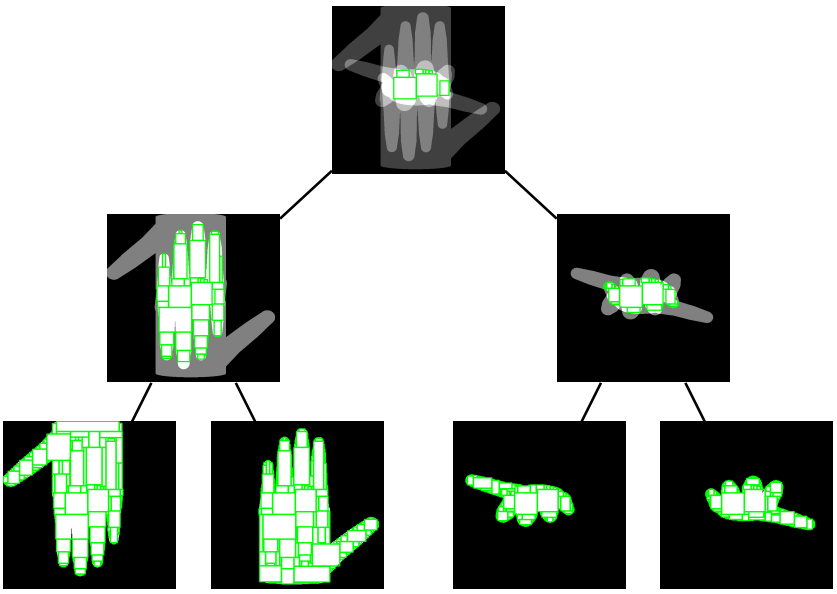

Figure 5: The figure shows the template hierarchy generated by our approach in Sec. 3.4. For the sake of clarity, only the rectangles approximating the foreground are shown.

can speed up the matching significantly by first matching with the intersection area, and only then continuing to match the remainders of the templates. This leads to a template hierarchy, the construction of which is guided by the area of intersection (see Fig 5 for an example).

3.4 Subdivision Criterion

In order to minimize matching effort, we subdivide the template set of size n into K subsets, such that the intersection area of the templates within each subset is maximized. The basic idea is to identify areas in the superposition that are covered by at least n/K templates. Additionally those areas should have the shape of axis-aligned rectangles. In other words, we try to subdivide the template such, that each subset can be covered by as few axis-aligned rectangles as possible. To achieve this, we define a distance measure between the templates silhouettes that is based on axis-aligned rectangles and not directly on the intersection itself.

First, we superimpose the templates $\mathcal{T} = \{T_j\}_{j=1\cdots m}$ and normalize the resulting image values to $[0, 1]$, which is denoted in the following by $H_{\mathcal{S}}$. Second, we apply our algorithm from Sec. 3.1 with $\tau = 1/K$ (see Eq. 4) to $H_{\mathcal{S}}$.

The result is the rectangle set $\mathcal{R}_{1\dots m}$. For each template T_i we compute the intersecting area v_i^j with each rectangle $R_i \in \mathcal{R}_{1\dots m}$:

$$\mathrm{v}_i^j = |T_j \cap R_i| \tag{10}$$

For each T_j we define a vector $\mathrm{v}_j = (\mathrm{v}_j^1, \cdots, \mathrm{v}_j^n),\ n = |\mathcal{R}_{1\dots m}|$. Next, we cluster the template set $\mathcal{T}$ into K disjoint subsets $\{\mathcal{T}_1 \dots \mathcal{T}_K\}$ by applying the batch neural gas clustering algorithm [Cottrell et al., 2005] to the set of vectors $\mathbf{v}_j$ using K prototypes. Performing many test runs showed that the clustering algorithm does its job well.

This method is applied recursively to $\mathcal{T} \dots \mathcal{T}_K$, until only one template is left per set, which yields our template tree. The root node is represented by the whole template set $\mathcal{T}$. Its children are $\{\mathcal{T}_1 \dots \mathcal{T}_K\}$ and so on.

For each node in the template tree, we compute a rectangle representation of the intersection of all templates contained in this node (with τ close to 1). Only the rectangle representation for each node is stored and used later for matching. Figure 5 illustrates the template hierarchy. In contrast to the independent approximation of each silhouette by rectangles (example in Fig. 3) most parts of the silhouette intersection areas are covered only once.

3.5 Template Tree Traversal

Based on this tree, constructed in the previous section, matching a template set $\mathcal{T}$ simply amounts to traversing the tree, which is described in more detail in the following.

We traverse the template tree in a cumulative way. While moving down the tree, more and more parts of the template(s) are matched to the input image. Let us denote the set of nodes in a template tree by $\{n_j\}_{j=1,\cdots,|\mathrm{Tree}|}$, the rectangle set at node n_j by $\mathcal{R}_j$, the parent of node n_j by $n_{p(j)}$ and the joint probability at n_j, evaluated at position p in an input image, by $P_j(\mathbf{p})$. Using Eq. 7, the joint probability at node n_j is:

$$P_j(\mathbf{p}) = P_{p(j)}(\mathbf{p}) + P_{S_j}(\mathbf{p}) \tag{11}$$

with

$$P_{\mathrm{root}}(\mathbf{p}) = P_{S_{\mathrm{root}}}(\mathbf{p}) \tag{12}$$

Obviously, the shape described by the cumulative rectangles from the root to the current node becomes more and more detailed. Thus, tree nodes representing template (sub-)sets closer to the object in the input image should produce higher match probabilities than other nodes. Without the normalization in Eq. 8 (replace $P_S(\mathbf{p})$ by $P_j(\mathbf{p})$), the highest matching scores would always be achieved in the root node, which is, clearly, not the desired result.

The reason is that at the root node, only the area shared by all templates is compared.

To get a more robust matching, we use multi-hypothesis tracking, i.e. we follow m paths from the root node to the leaves in parallel until we reach a leaf or P^N is below a predefined threshold. Contrary to intuition, it is not necessarily the best choice to descend into those branches that seem to offer the highest probability. We have tested three strategies: best-first search, breadth-first search and a custom traversal method, which we denote with *early-exit search* (testing the node with the lowest probability first). We experimentally found that, early-exit search works by far best. In other words, this strategy finds the correct template in much more frames than the other two strategies. We presume, the reason is that, if nodes with lowest probability are visited first, they are rejected early by the threshold test. Thus, more "space" is left for nodes that match better to the object in the input image.

In this section, we have presented a template hierarchy, exploiting the similarity between silhouette shapes is proposed. In the following, we will examine quality and run-time of our algorithm and compare it with a state-of-the-art approach.

4 Results

In our experiments, we have analyzed two aspects. The first one is the quality of our algorithm from Sec. 3.1 to compute a representation of templates by sets of axis-aligned rectangles. The second one measures and compares the quality and computation time of the template matching itself. In all experiments we have set the control parameter $\tau = 0.95$ (see Eq. 4).

4.1 Quality of our rectangle representation

First, we evaluated the quality of our approach approximating templates by axis-aligned rectangles. As quality measure we used the ratio of the benefit of the covering to the benefit of a perfect covering:

$$q = \frac{\sum_{R \in \mathcal{R}_S} F(R)}{F(\text{foreground}(T))} \tag{13}$$

In our experiments, we tried to cover a representative set of postures and orientations. The plot in Figure 6 shows the result. The images had a resolution of 1024×1024.

4.2 Evaluation of the matching quality

We compare our approach with a state-of-the-art approach proposed by Stenger et. al [Stenger et al., 2006], because our approach was inspired by theirs and

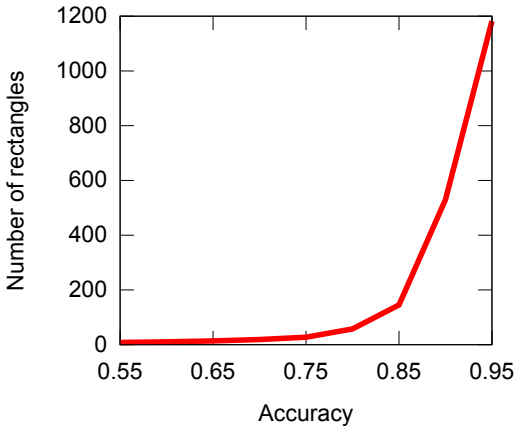

Figure 6: The plot shows the relation between the covering accuracy and the number of rectangles needed to approximate the silhouettes. The template silhouettes used are generated by an open hand with inplane-rotation (100 templates).

the application (hand tracking) is the same.

In the following, we will denote the algorithm from [Stenger et al., 2006] as *line-based matching (LBM)*, ours as *rectangle-based matching (RBM)*, and ours including the hierarchy *hierarchical RBM (HRBM)*. It is not quite fair to compare a hierarchical approach to non-hierarchical ones, but we add the results of the hierarchical match to our plots to analyze the potential of the hierarchy.

In the following, we will evaluate the difference between the methods with regard to resolution-independence, computation time, and accuracy. We generated templates with an artificial 3D hand model. We used the templates also as input images. There are two reasons to use such synthetic input datasets. First, we have the ground truth and, second, we can eliminate distracting influences like differences between hand model and real hand, image noise, bad illumination, and so on.

We generated three datasets for evaluation. Dataset 1, consisting of 1536 templates, is an open hand at different rotation angles. Dataset 2 is a pointing hand rendered at the same rotation angles as dataset 1. In dataset 3, consisting of 3072 templates, we used an open hand with abducting fingers. Additionally, for each position of the fingers, we rendered the model at different rotations.

First, we compared the matching quality of the three approaches. RBM and HRBM were evaluated at five different template approximation accu-

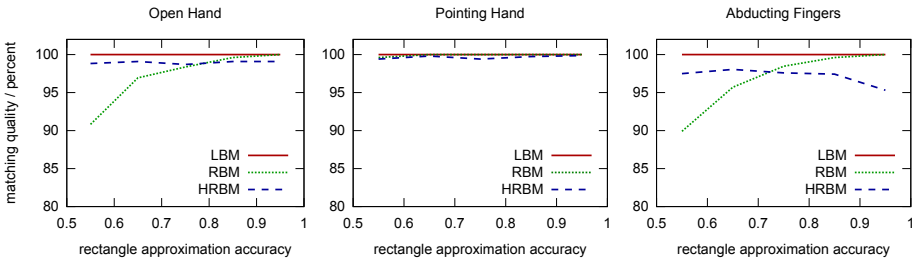

Figure 7: The matching quality of our two methods (RBM and HRBM) Just for reference, the quality of LBM [Stenger et al., 2006] is also plotted (LBM has no notion of template accuracy).

racies (see Eq. 13). We expected LBM to work best on the artificial datasets because, for each input image, there is one exactly matching template. For evaluation we used an input image resolution of 256×256 and compared each template at 5 different scalings (from 70×70 up to 200×200). All three approaches always found the correct location of the hand in the input image. Thus, for evaluation, we define the matching quality as the ratio of

$$\frac{\text{\# frames where the correct template was ranked among the top 10}}{\text{\# frames in the input sequence}} \tag{14}$$

at the correct position in the images. Please see Fig 7 for the results. The quality of RBM is as expected: the higher the rectangle approximation accuracy is, the higher the matching quality is. One notices that the matching quality in the *pointing hand* dataset is very high even at low rectangle approximation accuracy. We have taken a closer look at the datasets and found that the *pointing hand* has fewer similar 2D template shapes at different hand state parameters (i.e. less information loss during projection from 3D to 2D) The main reason for this is that the *pointing hand* shape is much more asymmetrically with respect to varying parameters as rotation and scale (for example a sphere is completely symmetric with respect to rotation). Figure 8 shows some examples. The results for HRBM at higher accuracy are slightly lower compared to RBM. The reason is that, in contrast to utilizing a "flat" set of templates, the matching algorithm never considers *all* templates because the tree traversal prunes large portions of the set of templates, which is the purpose of a hierarchy. Thus, utilizing any kind of hierarchical matching increases the likelihood of missing the best match, because that could happen to reside in a branch that was pruned. The quality of HRBM also depends on the clustering algorithm and the rectangles used to approximate the templates. This is the reason for the decreasing matching quality at the *abducting fingers* dataset at higher rectangle approximation accuracy.

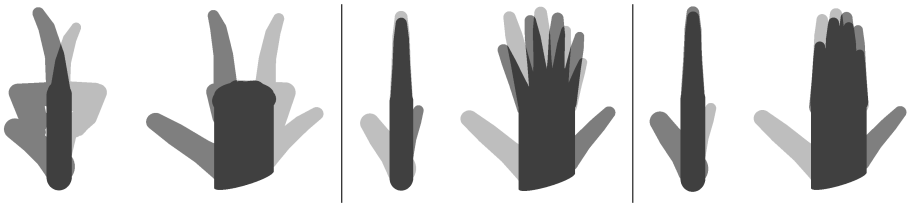

Figure 8: Two example configurations for the *pointing hand*, *open hand* and *abducting fingers* datasets. As you can see, the overlapping area in the *pointing hand* templates are smaller and thus, the probability for mismatching is lower too.

Second, we examined the dependence between the input image resolution and computation time. We have decided to use RBM and HRBM at a rectangle approximation accuracy of 0.75 because the plots in Figure 7 show that the matching quality is at most 3% lower than in the LBM method. We used input images at 5 different resolutions. We averaged the time to compute the joint probability for all frames at 49 positions each. The computation time of all three approaches are measured on a Intel Core2Duo 6700. The result is shown in Figure 9. Clearly, LBM's computation time depends linearly on the resolution, while our approaches exhibit almost constant time.

5 Conclusions

In this paper, we have presented a fast, adaptive template matching approach based on silhouette area matching. It works by approximating the template silhouettes by a set of axis-aligned rectangles. The accuracy of this representation can be adjusted by a parameter at this stage. We have also proposed a novel greedy algorithm to compute such a rectangle covering. Additionally, we have presented a template hierarchy, which utilizes our representation of the templates. This hierarchy reduces the computational complexity of the matching algorithm for a set of templates from linear to logarithmic time. Again, we would like to point out that our contributions constitute just one of the many pieces of a complete hand tracking system.

Overall, we need about $4.5\mu s$ on average to compare one template to one position in an input image at an arbitrary resolution (without using the hierarchy). This is about a factor 15 faster than the state-of-the-art approach from [Stenger et al., 2006] at a resolution of 1024×1024. Furthermore, the template representation is very memory efficient. For example, a template set consisting of 3000 templates needs less than 1.5 MByte storage space at

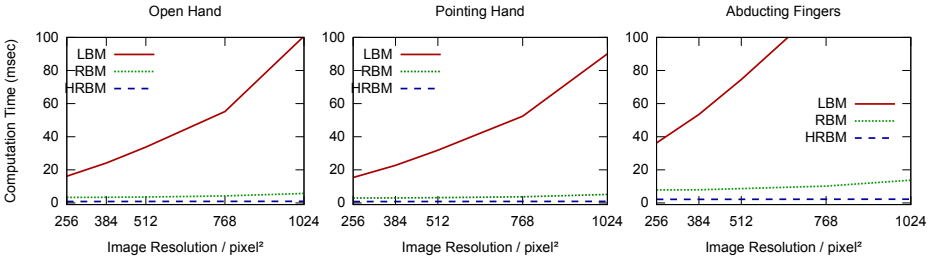

Figure 9: Each plot shows the average computation time for all three approaches: LBM [Stenger et al., 2006], RBM (our approach), HRBM (our approach incl. hierarchy). Clearly, our approaches are significantly faster and, even more important, resolution independent.

an accuracy of 0.75.

In the future, we plan to implement our approach in a massively parallel programming paradigm. Furthermore, we will extend our hierarchical approach to a random forest approach, which we expect to improve the template matching quality significantly. To get different classifiers at each node, one can choose a random subset instead of all covering templates to cluster a tree node for further subdivision.

References

[Amai et al., 2004] Amai, A., Shimada, N., and Shirai, Y. (2004). 3-d hand posture recognition by training contour variation. In *IEEE Conference on Automatic Face and Gesture Recognition*, pages 895–900.

[Cottrell et al., 2005] Cottrell, M., Hammer, B., Hasenfuß, A., and Villmann, T. (2005). Batch neural gas. In *5th Workshop On Self-Organizing Maps*.

[Crow, 1984] Crow, F. C. (1984). Summed-area tables for texture mapping. In *SIGGRAPH: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, volume 18.

[Heinrich-Litan and Lübecke, 2006] Heinrich-Litan, L. and Lübecke, M. E. (2006). Rectangle covers revisited computationally. In *ACM Journal of Experimental Algorithmics*, volume 11.

[Kato et al., 2006] Kato, M., Chen, Y.-W., and Xu, G. (2006). Articulated hand tracking by pca-ica approach. In *International Conference on Automatic Face and Gesture Recognition*, pages 329–334.

[Kumar and Ramesh, 1999] Kumar, V. A. and Ramesh, H. (1999). Covering rectilinear polygons with axis-parallel rectangles. In *Annual ACM Symposium on Theory of Computing*, pages 445–454.

[Lin et al., 2004] Lin, J. Y., Wu, Y., and Huang, T. S. (2004). 3D model-based hand tracking using stochastic direct search method. In *International Conference on Automatic Face and Gesture Recognition*, page 693.

[Liou et al., 1990] Liou, W., Tan, J. J.-M., and Lee, R. C. T. (1990). Minimum rectangular partition problem for simple rectilinear polygons. In *IEEE Transactions on Computer-Aided Design*, volume 9, pages 720–733.

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157.

[Nirei et al., 1996] Nirei, K., Saito, H., Mochimaru, M., and Ozawa, S. (1996). Human hand tracking from binocular image sequences. In *22th International Conference on Industrial Electronics, Control, and Instrumentation*, pages 297–302.

[O'Rourke and Tewari, 2001] O'Rourke, J. and Tewari, G. (2001). Partitioning orthogonal polygons into fat rectangles in polynomial time. In *In Proc. 13th Canadian Conference on Computational Geometry*, pages 97–100.

[Ouhaddi and Horain, 1999] Ouhaddi, H. and Horain, P. (1999). 3D hand gesture tracking by model registration. In *Workshop on Synthetic-Natural Hybrid Coding and Three Dimensional Imaging*, pages 70–73.

[Shimada et al., 2001] Shimada, N., Kimura, K., and Shirai, Y. (2001). Real-time 3-d hand posture estimation based on 2-d appearance retrieval using monocular camera. In *IEEE International Conference on Computer Vision*, page 23.

[Stenger et al., 2006] Stenger, B., Thayananthan, A., Torr, P. H. S., and Cipolla, R. (2006). Model-based hand tracking using a hierarchical bayesian filter. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 28, pages 1372–1384.

[Stenger, 2004] Stenger, B. D. R. (2004). Model-based hand tracking using a hierarchical bayesian filter. In *Dissertation submitted to the University of Cambridge*.

[Sudderth et al., 2004] Sudderth, E. B., Mandel, M. I., Freeman, W. T., and Willsky, A. S. (2004). Visual hand tracking using nonparametric belief propagation. In *IEEE CVPR Workshop on Generative Model Based Vision*, volume 12, page 189.

[Viola and Jones, 2001] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages I–511–I–518.

[Wu and Sahni, 1990] Wu, S. and Sahni, S. (1990). Covering rectilinear polygons by rectangles. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 9, pages 377–388.

[Wu et al., 2001] Wu, Y., Lin, J. Y., and Huang, T. S. (2001). Capturing natural hand articulation. In *International Conference on Computer Vision*, volume 2, pages 426–432.

[Zhou and Huang, 2003] Zhou, H. and Huang, T. (2003). Tracking articulated hand motion with eigen dynamics analysis. In *IEEE International Conference on Computer Vision*, volume 2, pages 1102–1109.

[Zhou and Huang, 2005] Zhou, H. and Huang, T. (2005). Okapi-chamfer matching for articulated object recognition. In *IEEE International Conference on Computer Vision*, volume 2, pages 1026–1033.